

SANDIA REPORT

2015-XXXX

Unlimited Release

Printed September 2015

A Brief Summary on Formalizing Parallel Tensor Distributions, Redistributions, and Algorithm Derivations

Martin D. Schatz, Robert van de Geijn, Taamra G. Kolda

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Approved for public release; further dissemination unlimited.



Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>



A Brief Summary on Formalizing Parallel Tensor Distributions, Redistributions, and Algorithm Derivations

Martin D. Schatz and Robert A. van de Geijn
Department of Computer Science
University of Texas
Austin, TX
martin.schatz@utexas.edu

Tamara G. Kolda
Sandia National Laboratories
Livermore, CA 94551
tgkolda@sandia.gov

Abstract

Large-scale datasets in computational chemistry typically require distributed-memory parallel methods to perform a special operation known as tensor contraction. Tensors are multidimensional arrays, and a tensor contraction is akin to matrix multiplication with special types of permutations. Creating an efficient algorithm and optimized implementation in this domain is complex, tedious, and error-prone. To address this, we develop a notation to express data distributions so that we can apply use automated methods to find optimized implementations for tensor contractions. We consider the spin-adapted coupled cluster singles and doubles method from computational chemistry and use our methodology to produce an efficient implementation. Experiments performed on the IBM Blue Gene/Q and Cray XC30 demonstrate impact both improved performance and reduced memory consumption.

Acknowledgment

This is a summary of the first authors' Ph.D. thesis [\[21\]](#), which was supported by Sandia's Campus Executive program. It is the final report for LDRD Project entitled, "A Domain-Specific Language for Distributed Tensor Computations" (Project # 181062, Proposal # 15-1176).

Contents

Nomenclature	6
1 Introduction	7
2 Detailed Description of Experiment/Method	8
2.1 Preliminaries	8
2.2 Notation for Data Distributions	8
2.3 Data Redistributions	10
2.4 Algorithm Derivation	10
2.5 Blocking	13
3 Results	14
3.1 IBM Blue Gene/Q Experiments	14
3.2 Cray XC30 Experiments	14
3.3 Weak Scalability Experiments	17
4 Anticipated Impact	21
4.1 Symmetry & Sparsity	21
4.2 Additional Families of Algorithms	21
4.3 Additional Data Distributions	21
4.4 Additional Tensor Operations	21
4.5 Aiding Automated Tools	22
5 Conclusion	23

Figures

1	Graphical depiction of the matrix \mathbf{A} distributed according to different tensor distributions represented in the defined notation. The top-left entry of every container corresponds to the process's location within the mesh.	9
2	Performance results on IBM Blue Gene/Q architecture comparing to CTF with different numbers of compute nodes. The top of each graph represents the theoretical peak for the configuration. Dashed vertical lines indicate the percentage of total memory consumed by inputs.	15
3	Performance results on Cray XC30 architecture comparing to CTF with different numbers of compute nodes. The top of each graph represents the theoretical peak of the configuration. Dashed vertical lines indicate the percentage of total memory consumed by inputs.	16
4	Timing results on Cray XC30 architecture comparing to NWChem with different numbers of compute nodes (lower is better). Dashed vertical lines indicate the percentage of total memory consumed by inputs.	18
5	Weak scalability of the ROTE-based implementations on different architectures. Weak scalability measurements were performed with problems whose inputs consume approximately fifty percent of available memory. The top of the graph corresponds to the theoretical peak of each configuration tested. ...	19

Nomenclature

CCSD coupled cluster single and doubles

CCSDT coupled cluster singles, doubles, and triples

CTF Cyclops Tensor Framework (software package)

DxTer Prototype system for design by transformation

NWChem NorthWest computational Chemistry (software package)

ROTE Redistribution Operations and Tensor Expressions

1 Introduction

Quantum mechanical models have proven to be highly accurate for physical systems. These systems are described by an associated function, called the system’s *wave function*. Different theories of approximating the true wave function exist [24, 6, 25, 9]. One such theory, known as coupled cluster theory [6], has been shown to accurately model chemical systems while retaining desired properties of the model. In coupled cluster theory, a set of *cluster operators* are defined that provide an exact representation of the wave function associated with the system of interest. We focus a method that accounts for both single and double excitations, referred to as the “coupled cluster singles and doubles (CCSD)” method [19]. Many other methods exist, such as the “coupled cluster singles, doubles, and triples (CCSDT)” method [17]. For each such method, an associated set of equations defines how to compute the coefficients that correspond to the appropriate approximate wave function. Depending on the particular method, the resulting set of equations can have different computational characteristics. Further approximations can be made or techniques can be used to reduce the complexity of the resulting equations or factorization process, and *spin-adaptation* refers to a technique that produces sets of equations that can be efficiently computed [16]. We focus on the spin-adapted CCSD in this work, but the methods we develop are more general.

The key operation in any of these methods is a series of *tensor contractions* and typically require distributed-memory parallel solutions to perform the computation because the data is so large. Tensors are multidimensional arrays and can be considered generalizations of matrices. The order of a tensor is the number of ways or modes that it represents. We say a tensor is higher-order if its order is greater than two. Tensor contractions are a generalization of matrix-matrix multiplication. The challenge in developing efficient implementations for applications based on tensor contractions stems from the increased number of data distributions and algorithmic variants available. The simplest approach to computing a tensor contraction is to permute the data and perform a matrix-matrix multiplication, leveraging previous work on parallelizing that operation. Unfortunately, this reduces the opportunities for improving performance and reducing memory requirements.

The goal of this work is to optimize a series of tensor contractions using automated tools. To do this, we have to consider not only how to optimize the individual contractions, but also consider the conjunctions where the output of one contraction is the input the next. To automate this process, we need a well-defined language that expresses the distributions and redistributions of the tensor objects.

We can borrow from the ideas from matrix-matrix multiplication [22, 27, 28, 20] because it is closely related to tensor contraction. We develop a notation to encode tensor operations and distributions on a multidimensional processing mesh. It can be used to explain the data redistributions that occur from efficient collective communications. This combination leads to a systematic approach to derive algorithms for a series of tensor contraction operations, making the algorithmic impact of the choices clear and simplifying the decision-making in designing optimization algorithms. Additionally, decisions potentially error-prone hand-coded optimization can be automated.

2 Detailed Description of Experiment/Method

Here, we briefly describe the notation developed for data distributions and redistributions as well as the systematic algorithm derivation procedure exposed by the notation. Details of the notation as well as the derivation procedure can be found in [21].

2.1 Preliminaries

We use boldface capital letters to refer to tensors (\mathbf{A} , \mathbf{B} , \mathbf{C}). The order of a data tensor is denoted M . If necessary, a parenthesized superscript is used to differentiate between the tensors being considered; e.g., $M^{(\mathbf{A})}$ indicates the order of the tensor \mathbf{A} .

We use $\mathbf{I} = (I_0, \dots, I_{M-1})$ to refer to the size of an order- M tensor. When referencing an element of the order- M tensor \mathbf{A} , we specify its location in \mathbf{A} with an M -tuple, or *multiindex*, $\mathbf{i} = (i_0, \dots, i_{M-1})$ with entries corresponding to the element's index in each mode of the tensor. Again, sizes and multiindices of a specific tensor are distinguished by a parenthesized superscript if necessary.

2.2 Notation for Data Distributions

In our notation, indices of each tensor mode are element-cyclic distributed on a subset of modes of an order- N processing mesh \mathbf{G} . The combination of assigned indices determines the set of elements assigned to each process. In our notation, for each tensor mode, we associate an ordered set of processing mesh modes called a *mode distribution*.

The processing mesh modes comprising each mode distribution indicate the modes along which the corresponding tensor mode indices are elemental-cyclic distributed. A *tensor distribution* then becomes nothing more than the collection of mode distributions associated with each modes of the tensor to be distributed. As a shorthand, to represent the order- M tensor \mathbf{A} distributed with mode u distributed according to the mode distribution $\mathcal{D}^{(u)}$, we write $\mathbf{A} [\mathcal{D}^{(0)}, \dots, \mathcal{D}^{(M)}]$.

For instance,

$$\mathbf{A} [(0), (1)]$$

represents an order-2 tensor \mathbf{A} distributed such that the mode-0 indices of \mathbf{A} are elemental-cyclic distributed among mode 0 of the processing mesh, and the mode-1 indices of \mathbf{A} are elemental-cyclic distributed among mode 1 of the processing mesh.

The order of elements in each mode distribution indicates the order of precedence each processing mesh mode is given when determining the set of mode indices assigned to a given process. For instance, the mode distribution $(0, 1)$ indicates that the corresponding mode indices should be distributed elemental-cyclic along mode mode 0 and mode 1 of the

(0, 0)	(0, 1)	(0, 2)
$a_{0,0}$ $a_{0,3}$ \dots	$a_{0,1}$ $a_{0,4}$ \dots	$a_{0,2}$ $a_{0,5}$ \dots
$a_{2,0}$ $a_{2,3}$ \dots	$a_{2,1}$ $a_{2,4}$ \dots	$a_{2,2}$ $a_{2,5}$ \dots
\vdots \vdots \ddots	\vdots \vdots \vdots	\vdots \vdots \ddots
(1, 0)	(1, 1)	(1, 2)
$a_{1,0}$ $a_{1,3}$ \dots	$a_{1,1}$ $a_{1,4}$ \dots	$a_{1,2}$ $a_{1,5}$ \dots
$a_{3,0}$ $a_{3,3}$ \dots	$a_{3,1}$ $a_{3,4}$ \dots	$a_{3,2}$ $a_{3,5}$ \dots
\vdots \vdots \ddots	\vdots \vdots \ddots	\vdots \vdots \ddots

(a) $\mathbf{A}[(0), (1)]$

(0, 0)	(0, 1)	(0, 2)
$a_{0,0}$ $a_{0,6}$ \dots	$a_{0,1}$ $a_{0,7}$ \dots	$a_{0,2}$ $a_{0,8}$ \dots
$a_{1,0}$ $a_{1,6}$ \dots	$a_{1,1}$ $a_{1,7}$ \dots	$a_{1,2}$ $a_{1,8}$ \dots
\vdots \vdots \ddots	\vdots \vdots \ddots	\vdots \vdots \ddots
(1, 0)	(1, 1)	(1, 2)
$a_{0,3}$ $a_{0,9}$ \dots	$a_{1,4}$ $a_{0,10}$ \dots	$a_{0,5}$ $a_{0,11}$ \dots
$a_{1,3}$ $a_{1,9}$ \dots	$a_{1,4}$ $a_{1,10}$ \dots	$a_{1,5}$ $a_{1,11}$ \dots
\vdots \vdots \ddots	\vdots \vdots \ddots	\vdots \vdots \ddots

(b) $\mathbf{A}[(\cdot), (1, 0)]$

Figure 1: Graphical depiction of the matrix \mathbf{A} distributed according to different tensor distributions represented in the defined notation. The top-left entry of every container corresponds to the process's location within the mesh.

processing mesh. Further, this mode distribution indicates that a process's location in mode 0 of the mesh should take higher precedence than its location in mode 1 of the mesh when determining the set of indices assigned to it.

When restricted to an order-2 processing mesh, the mode distribution $(0, 1)$ represents an elemental-cyclic distribution of indices among processes as if the processes were arranged as an order-1 processing mesh and placed in *column*-major order. Contrast this to the mode distribution $(1, 0)$ that similarly distributes indices among both modes of the processing mesh, except that the processes are arranged in a *row*-major order.

Two examples of data distributions using our notation are illustrated in Figure 1. Further details on how distributions are defined are given in [21].

2.3 Data Redistributions

Specific data redistributions are associated with each collective communications in distributed-memory parallel matrix computations. We can express the change using our notation and define rules to say the costs and possible implementation variations for each redistribution. If a required redistribution does not directly map to one of the defined rules, then it can be implemented by composing redistribution rules together. In [21], a procedure is given for systematically decomposing redistributions along these lines.

2.4 Algorithm Derivation

For distributed-memory parallel matrix-matrix multiplication, the stationary family of algorithms have been shown to lead to high-performance implementations [23]. These algorithms assume one operand is significantly larger than the others and avoid communicating this operand; the other operands are allowed to be communicated because they are smaller. Nevertheless, the developer is still required to make choices on how to distribute the data. in this subsection, we show an example of how our notation exposes the fixed parameters and options in a tensor contractions. Although we show just a single contraction, this general procedure can be applied to a series of tensor contractions. Additional details of this procedure can be found in [21]

Consider the tensor contraction

$$\mathbf{C}^{\alpha\beta\eta\iota} = \mathbf{A}^{\alpha\gamma\iota\kappa} \mathbf{B}^{\beta\gamma\eta\kappa} + \mathbf{C}^{\alpha\beta\eta\iota}$$

where \mathbf{C} , \mathbf{A} and \mathbf{B} are conformally sized. For this example, our goal is to derive a stationary- \mathbf{C} algorithm that computes the above expression without communicating \mathbf{C} . We assume an order-4 processing mesh \mathbf{G} of size $\mathbf{P} = P_0 \times P_1 \times P_2 \times P_3$.

We want to derive an algorithm that matches the template

$$\begin{aligned}
1. \quad & \mathbf{A} [?, ?, ?, ?] \leftarrow \mathbf{A} [?, ?, ?, ?] && \text{(redistribute } \mathbf{A} \text{)} \\
2. \quad & \mathbf{B} [?, ?, ?, ?] \leftarrow \mathbf{B} [?, ?, ?, ?] && \text{(redistribute } \mathbf{B} \text{)} \\
3. \quad & \mathbf{C} [?, ?, ?, ?] \leftarrow \mathbf{C} [?, ?, ?, ?] && \text{(redistribute } \mathbf{C} \text{)} \\
4. \quad & \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [?, ?, ?, ?, ?, ?] = \widehat{\sum_{\gamma\kappa}} \mathbf{A}^{\alpha\gamma\iota\kappa} [?, ?, ?, ?] \mathbf{B}^{\beta\gamma\eta\kappa} [?, ?, ?, ?] && \text{(local contraction)} \quad (1) \\
5. \quad & \mathbf{C}^{\alpha\beta\eta\iota} [?, ?, ?, ?] += \widetilde{\sum_{\gamma'\kappa'}} \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [?, ?, ?, ?, ?, ?] && \text{(global reduction)} \\
6. \quad & \mathbf{C} [?, ?, ?, ?] \leftarrow \mathbf{C} [?, ?, ?, ?]. && \text{(redistribute } \mathbf{C} \text{)}
\end{aligned}$$

where our goal is to fill in the unknown entries denoted with “?”. Steps 1-3 and 6 correspond to redistributions of data, and Steps 4-5 perform the necessary computations. The symbols $\widehat{\sum}$ and $\widetilde{\sum}$ denote local and global summations, respectively. We use knowledge about our problem specification to fill in details of the template and potentially remove unnecessary steps. For instance, using a stationary- \mathbf{C} algorithm implies that nothing will occur in Steps 3 and 6. We use the template as given for a starting point as it can be used to derive all the stationary variants.

The derivation procedure is not dependent on a choice of initial distribution, however, we know that tensor distributions that involve all modes of the processing mesh do not implicitly replicate data. Let us derive an algorithm that assumes each tensor is initially distributed such that there is no replication of data among processes. A convenient form for this is to assume that the incoming tensors \mathbf{A} , \mathbf{B} , and \mathbf{C} are initially distributed as

$$\mathbf{A} [(0), (1), (2), (3)], \mathbf{B} [(0), (1), (2), (3)], \text{ and } \mathbf{C} [(0), (1), (2), (3)];$$

in other words, via an elemental-cyclic distribution. Combining this with the stationary- \mathbf{C} assumption, our template now becomes

$$\begin{aligned}
1. \quad & \mathbf{A} [?, ?, ?, ?] \leftarrow \mathbf{A} [(0), (1), (2), (3)] \\
2. \quad & \mathbf{B} [?, ?, ?, ?] \leftarrow \mathbf{B} [(0), (1), (2), (3)] \\
4. \quad & \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [?, ?, ?, ?, ?, ?] = \widehat{\sum_{\gamma\kappa}} \mathbf{A}^{\alpha\gamma\iota\kappa} [?, ?, ?, ?] \mathbf{B}^{\beta\gamma\eta\kappa} [?, ?, ?, ?] \\
5. \quad & \mathbf{C}^{\alpha\beta\eta\iota} [(0), (1), (2), (3)] += \widetilde{\sum_{\gamma'\kappa'}} \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [?, ?, ?, ?, ?, ?].
\end{aligned} \quad (2)$$

Since Step 5 in (2) requires a global communication, which represents overhead, we want to eliminate this step. To do so, we distribute the paired modes of \mathbf{T} and \mathbf{C} similarly,

ensuring no communication. Hence, the procedure is updated as

$$\begin{aligned}
1. \quad & \mathbf{A} [?, ?, ?, ?] \leftarrow \mathbf{A} [(0), (1), (2), (3)] \\
2. \quad & \mathbf{B} [?, ?, ?, ?] \leftarrow \mathbf{B} [(0), (1), (2), (3)] \\
4. \quad & \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [(0), (1), (2), (3), ?, ?] = \widehat{\sum_{\gamma\kappa}} \mathbf{A}^{\alpha\gamma\iota\kappa} [?, ?, ?, ?] \mathbf{B}^{\beta\gamma\eta\kappa} [?, ?, ?, ?] \\
5. \quad & \mathbf{C}^{\alpha\beta\eta\iota} [(0), (1), (2), (3)] += \widetilde{\sum_{\gamma'\kappa'}} \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [(0), (1), (2), (3), ?, ?].
\end{aligned} \tag{3}$$

Likewise, Step 4 in (3) corresponds to a local tensor contraction and therefore all paired modes must be distributed similarly to ensure the local computation succeeds. This yields

$$\begin{aligned}
1. \quad & \mathbf{A} [(0), ?, (3), ?] \leftarrow \mathbf{A} [(0), (1), (2), (3)] \\
2. \quad & \mathbf{B} [(1), ?, (2), ?] \leftarrow \mathbf{B} [(0), (1), (2), (3)] \\
4. \quad & \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [(0), (1), (2), (3), ?, ?] = \widehat{\sum_{\gamma\kappa}} \mathbf{A}^{\alpha\gamma\iota\kappa} [(0), ?, (3), ?] \mathbf{B}^{\beta\gamma\eta\kappa} [(1), ?, (2), ?] \\
5. \quad & \mathbf{C}^{\alpha\beta\eta\iota} [(0), (1), (2), (3)] += \widetilde{\sum_{\gamma'\kappa'}} \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [(0), (1), (2), (3), ?, ?].
\end{aligned} \tag{4}$$

Considering the determined tensor mode distributions along with the fact that we have assumed an order-4 processing mesh reveals that the only valid assignment for the remaining tensor mode distributions is the empty set (indicating a duplication of the tensor-mode indices assigned to processes). Propagating this information leads to the template

$$\begin{aligned}
1. \quad & \mathbf{A} [(0), (), (3), ()] \leftarrow \mathbf{A} [(0), (1), (2), (3)] \\
2. \quad & \mathbf{B} [(1), (), (2), ()] \leftarrow \mathbf{B} [(0), (1), (2), (3)] \\
4. \quad & \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [(0), (1), (2), (3), (), ()] = \widehat{\sum_{\gamma\kappa}} \mathbf{A}^{\alpha\gamma\iota\kappa} [(0), (), (3), ()] \mathbf{B}^{\beta\gamma\eta\kappa} [(1), (), (2), ()] \\
5. \quad & \mathbf{C}^{\alpha\beta\eta\iota} [(0), (1), (2), (3)] += \widetilde{\sum_{\gamma'\kappa'}} \mathbf{T}^{\alpha\beta\eta\iota\gamma'\kappa'} [(0), (1), (2), (3), (), ()].
\end{aligned} \tag{5}$$

At this point, we have defined all unknowns and have arrived at a valid algorithm. However, notice that Step 5 in (5) is performing a global reduction over no processing mesh modes and is here an assignment rather than a summation. Therefore, by replacing \mathbf{C} with \mathbf{T} in Step 4, we can remove Step 5 from our derived algorithm. This leads us to our final template given by

$$\begin{aligned}
1. \quad & \mathbf{A} [(0), (), (3), ()] \leftarrow \mathbf{A} [(0), (1), (2), (3)] \\
2. \quad & \mathbf{B} [(1), (), (2), ()] \leftarrow \mathbf{B} [(0), (1), (2), (3)] \\
4. \quad & \mathbf{C}^{\alpha\beta\eta\iota} [(0), (1), (2), (3)] = \widehat{\sum_{\gamma\kappa}} \mathbf{A}^{\alpha\gamma\iota\kappa} [(0), (), (3), ()] \mathbf{B}^{\beta\gamma\eta\kappa} [(1), (), (2), ()].
\end{aligned} \tag{6}$$

As mentioned, there may be places where a choice must be made that can impact performance. This is where expert knowledge is still required to determine the optimal path of redistributions (minimal cost or storage). However, the derivation procedure has significantly reduced the number of possibilities that need to be considered.

2.5 Blocking

In Step 4 of (6), the distributions of \mathbf{A} and \mathbf{B} indicate a replication of data among processes as some modes of \mathbf{G} are not used in the associated tensor distribution. As we increase the number of processes involved, so does the amount of data replication, with associated memory requirements for the duplicated data. One straight-forward approach to curb this effect is to block the overall tensor contraction into a series of smaller, *block* tensor contractions to which we can apply the same derivation process.

Depending on how we choose the size of each block, different characteristics of the execution will be observed. If a large block size is chosen, a large amount of data is required to perform the computation. If a small block size is chosen, reducing the amount of storage required, a higher communication cost is predicted due to the increased number of communications being performed (one round for each block). It is this trade-off that we must recognize and appropriately analyze to ensure an efficient implementation is created.

Since we are dealing with higher-order tensors, determining the correct modes to block may seem daunting at first. Observe that the source of replicated data in (6) are the distributed tensors $\mathbf{A}[(0), (), (3), ()]$ and $\mathbf{B}[(1), (), (2), ()]$. Comparing the distributions of these tensors to the initial distributions, we see that the replication of data originates from the redistribution of the tensor modes involved in the summation. If we increase the number of processes in our processing mesh, then replication will occur due to the redistribution of these tensor modes. By blocking along these modes, we can mitigate this effect.

Generalizing this observation, notice that the tensor modes we should block along in (6) correspond to the modes that are unpaired with our stationary tensor \mathbf{C} . In fact, for all stationary variants, the observation that replication occurs due to the redistribution of tensor modes not paired with the stationary operand and should be blocked holds. This reasoning provides an expert with a simple way of determining along which tensor modes to introduce blocking, thereby mitigating the increased storage effect as the size of the processing mesh increases.

3 Results

The ideas developed in this work were encoded into the prototype system for *design by transformation* (DxTer) [15, 14] for generating efficient implementations using our Redistribution Operations and Tensor Expressions (ROTE) software. We applied this to the spin-adapted CCSD method from computational chemistry. Experiments were performed on both the Blue Gene/Q and Cray XC30 computing architectures and performance was compared to the Cyclops Tensor Framework (CTF) [28] and the NorthWest computational Chemistry (NWChem) [29] software packages, both state-of-the-art libraries for these computations.

3.1 IBM Blue Gene/Q Experiments

Comparison with CTF. In Figure 2, we show experimental results comparing ROTE and CTF on 32 and 512 nodes (512 and 8192 cores, respectively) on the IBM Blue Gene/Q system. The results show that the ROTE-based implementations achieve performance that is as good or better than CTF in terms of GFLOPS. Additionally, the ROTE-based scales to larger problem size because it is more efficient in its memory usage.

For the 32-node case, a configuration of eight MPI ranks per node and eight OpenMP threads per MPI rank achieved the best results for each library. For the 512-node case, a configuration of one MPI rank and sixty-four OpenMP threads per rank achieved the best performance for both libraries. CTF handles its processor configuration internally. For ROTE, we used a processing mesh of size $4 \times 4 \times 4 \times 4$ for the 32-node experiment and of size $2 \times 4 \times 8 \times 8$ for the 512-node experiment. Note that in the 32-node experiment, all grid modes have the same dimension meaning that many redistributions relying on all-to-all collectives can be implemented in terms of permutation collectives.

For experiments on 512 nodes, we hypothesize that the reason DxTer did not pick a processing mesh of size $8 \times 8 \times 8 \times 8$ is due to the allgather redistributions. Specifically, data duplication occurs on modes 0 and 1 of the processing mesh. In the case of the $8 \times 8 \times 8 \times 8$ configuration, this means that sixty-four processes are involved in the redistributions, whereas in the $2 \times 4 \times 8 \times 8$ configuration, only eight are. This is an example of where the tradeoff between performance of different transformations and collective communications must be considered to achieve a higher performing implementation; in the $2 \times 4 \times 8 \times 8$ case, we limit the opportunities to implement an all-to-all collective as a permutation collective, but reduce the cost of allgather collectives, whereas the $8 \times 8 \times 8 \times 8$ case has the opposite property.

3.2 Cray XC30 Experiments

Comparison with CTF. In Figure 3, we compare ROTE and CTF using 32 and 512 nodes (768 and 12288 cores, respectively) on the Cray XC30 architecture. A node configuration of

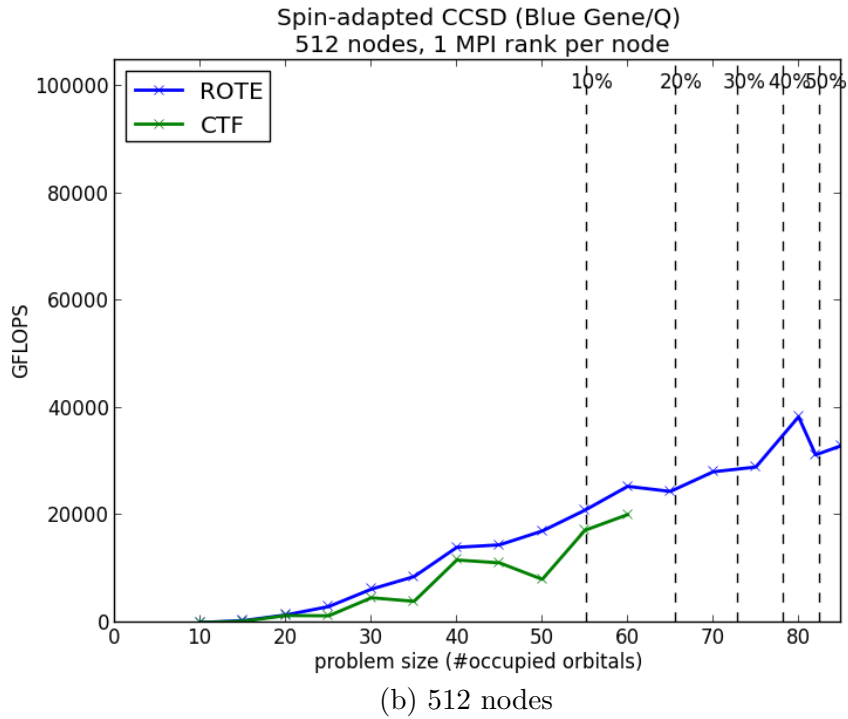
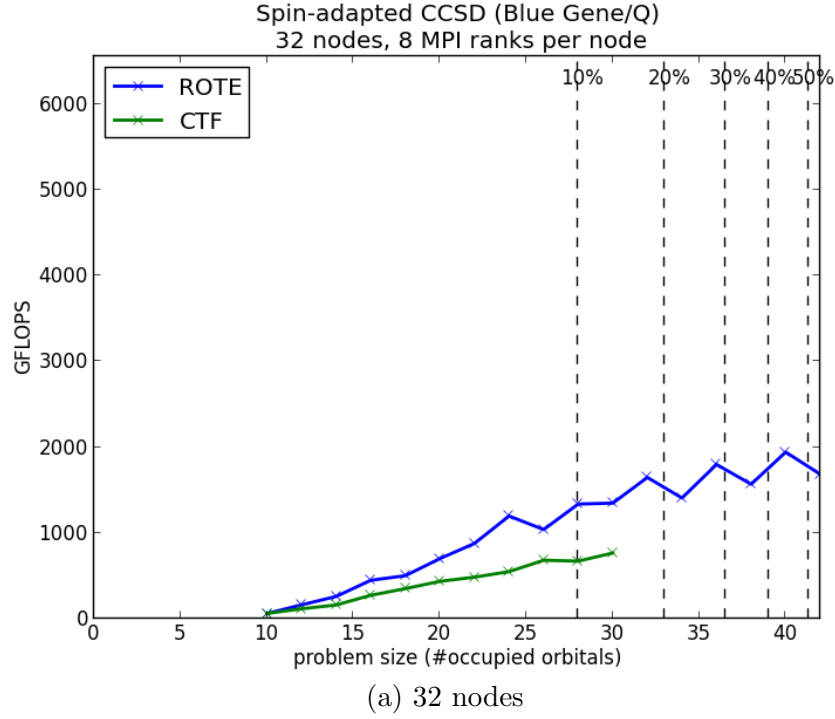


Figure 2: Performance results on IBM Blue Gene/Q architecture comparing to CTF with different numbers of compute nodes. The top of each graph represents the theoretical peak for the configuration. Dashed vertical lines indicate the percentage of total memory consumed by inputs.

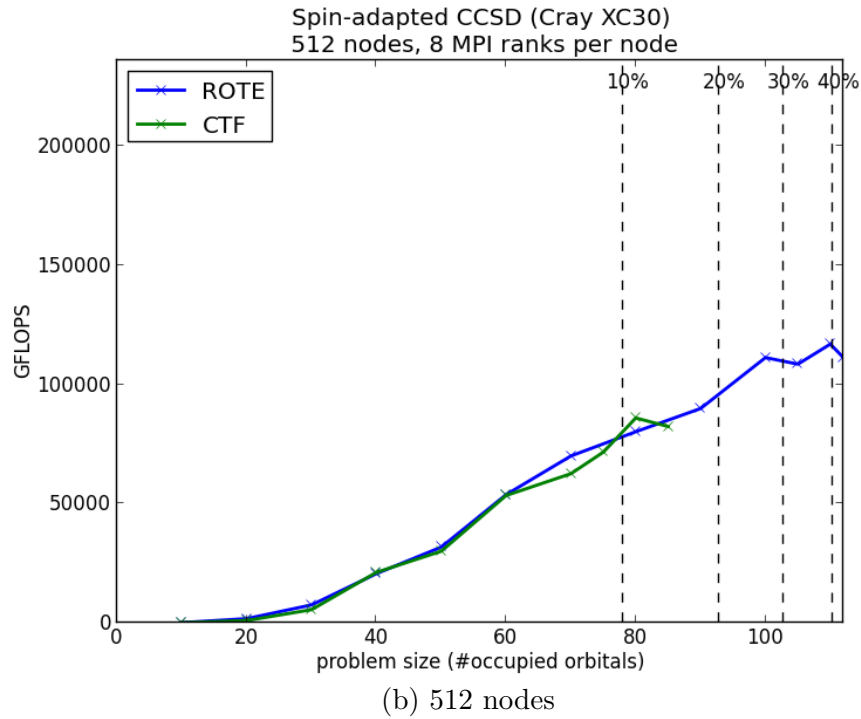
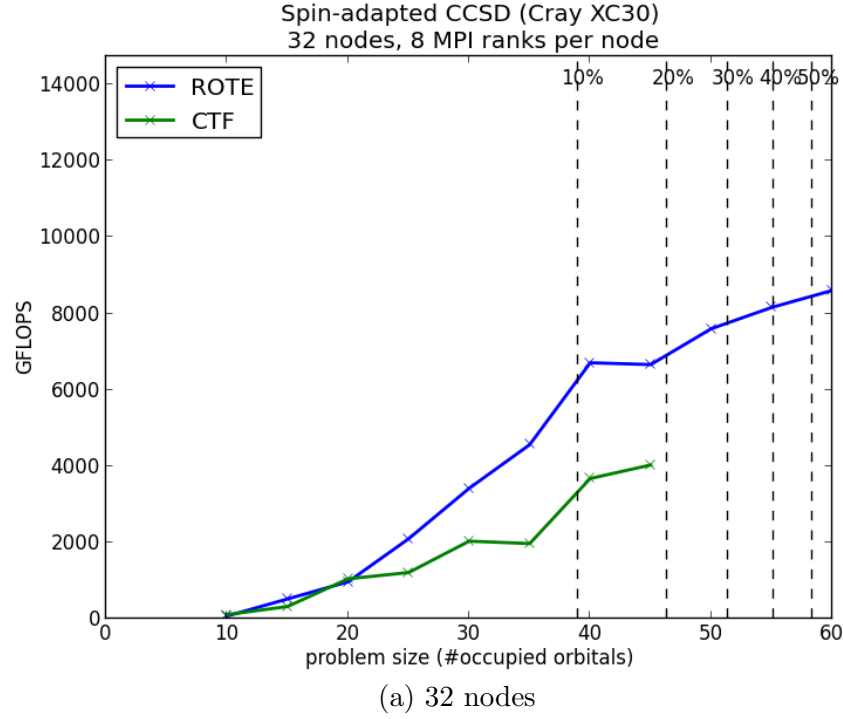


Figure 3: Performance results on Cray XC30 architecture comparing to CTF with different numbers of compute nodes. The top of each graph represents the theoretical peak of the configuration. Dashed vertical lines indicate the percentage of total memory consumed by inputs.

eight MPI ranks per node and three OpenMP threads per MPI rank achieved the best results for both ROTE and CTF in both the 32- and 512-node experiments. For ROTE, we use a processing mesh of size $4 \times 4 \times 4 \times 4$ for 32 nodes and of size $8 \times 8 \times 8 \times 8$ for 512 nodes. For 32 nodes, ROTE generally outperforms CTF, but for 512 nodes, is sometimes a little better. The differences are not as extreme as for the IBM architecture. We suspect that this is due to the relatively faster communication network of the Cray XC30 architecture that allows differences in communication performance to be hidden [30, 18].

Comparison with NWChem. In Figure 4, we compare runtimes for ROTE and NWChem using 32 and 512 nodes (768 and 12288 cores, respectively) on the Cray XC30 architecture. We compare runtimes rather than FLOPS because each implementation is performing slightly different computations.

As with CTF, the node configuration is not a parameter to NWChem; therefore, we merely compare the best performance of ROTE-based implementations to the best achieved by NWChem. When performing these experiments, we discovered a bug in an NWChem implementation that utilized asynchronous communications (theoretically optimizing the encountered communications) for the Cray XC30 architecture rendering it unusable for comparison. Instead, we compared against an implementation of NWChem (6.3) that did not use this feature (similar to the stable implementation of NWChem) as recommended to us by experts associated with NWChem.

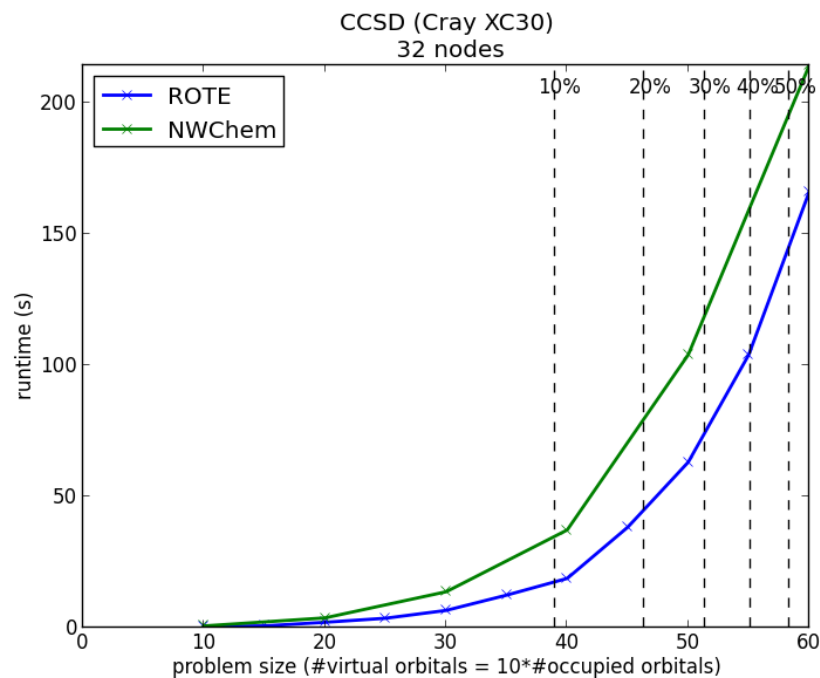
NWChem relies on an approach to computing the dominant term of CCSD that exploits symmetry and therefore significantly reduces the overall storage requirement. As this approach is not currently available in ROTE, we cannot solve problems that are as large. Therefore, we stop the comparison at the largest problem size ROTE was able to compute. Additionally, the exact equations being computed differ significantly from each package; however, chemically they represent methods with similar computational complexity.

Considering the assumptions about these experiments, we see that although ROTE can only solve smaller problems, it does outperform NWChem.

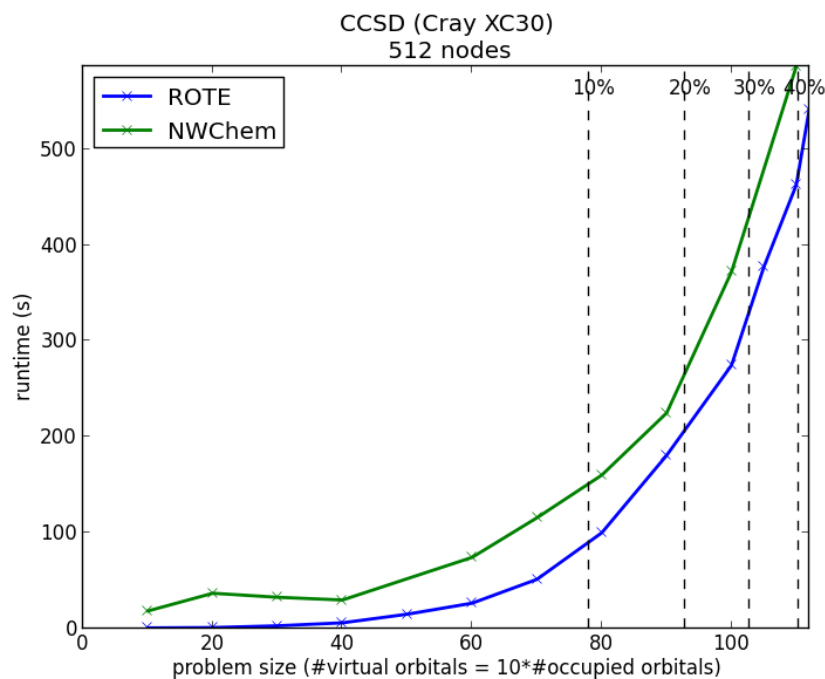
3.3 Weak Scalability Experiments

We perform weak scalability experiments, meaning that we increase the problem size in proportion to the number of processing elements. For this metric, a perfectly scalable implementation would maintain its performance (e.g., in terms of FLOPS or fraction of peak performance). In Figure 5, we show weak scalability results for ROTE on the IBM and Cray systems.

For each data point, we selected a problem that was large enough so that approximately fifty percent of the available storage was reserved for inputs of the CCSD application. The results show good scaling. For the Blue Gene/Q, the fraction of peak performance is nearly constant. For the Cray XC30, the performance dips slightly as the number of nodes increases.

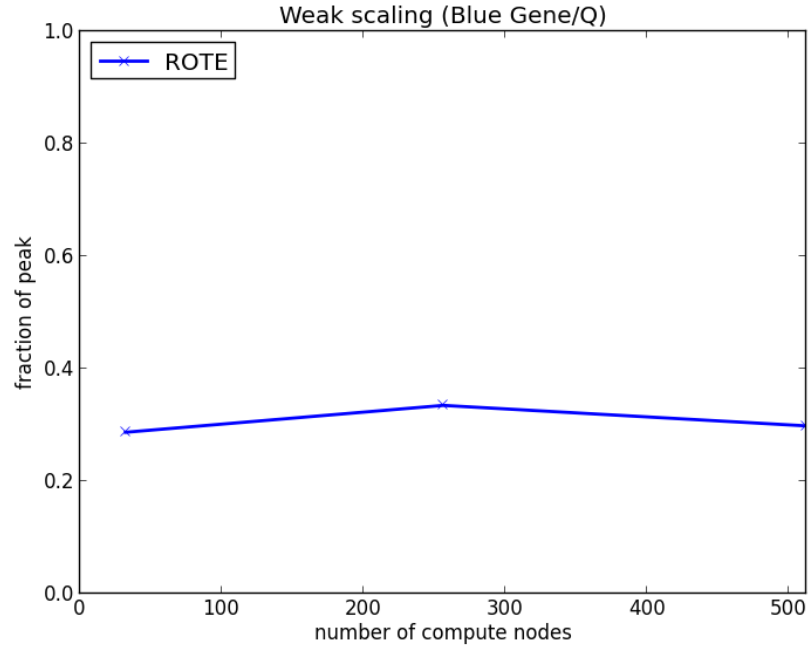


(a) 32 nodes

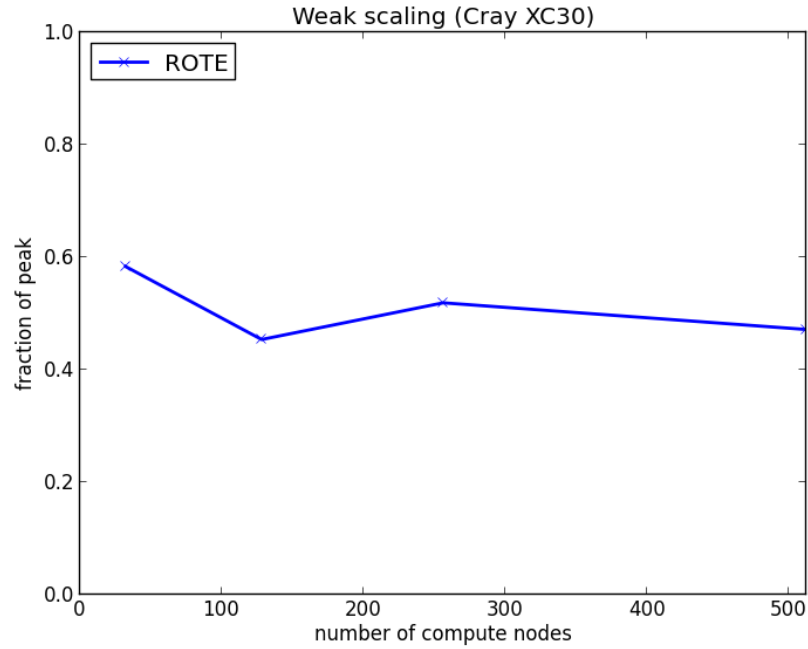


(b) 512 nodes

Figure 4: Timing results on Cray XC30 architecture comparing to NWChem with different numbers of compute nodes (lower is better). Dashed vertical lines indicate the percentage of total memory consumed by inputs.



(a) IBM BlueGene/Q



(b) Cray XC30

Figure 5: Weak scalability of the ROTE-based implementations on different architectures. Weak scalability measurements were performed with problems whose inputs consume approximately fifty percent of available memory. The top of the graph corresponds to the theoretical peak of each configuration tested.

Overall, this provides some evidence of relatively good weak scalability.

We mention here we may use a different optimized implementation for each number of processors. We expect that if the same implementation is chosen both 32 and 512 nodes, then performance would degrade as the relationship between processing mesh configuration and the selected collectives may be less advantageous.

4 Anticipated Impact

We anticipate that this work is both advancing the state of the art for distributed-memory parallel computational chemistry methods, as well as providing a way to reason about implementing distributed-memory tensor operations. To that end, we discuss topics relating to more general tensor operations.

4.1 Symmetry & Sparsity

Many applications in computational chemistry involve symmetric tensors [17, 12]. Exploiting symmetry can greatly reduce the storage and computational requirements, so incorporating symmetry into the notation and theory is a topic for future research. In addition to symmetry, some methods rely on sparsity in the tensors to reduce the overall computational cost and storage requirements [8, 10, 13]. So far we have focused on dense tensors because each process can be assigned a predictable and structured set of elements that maintains balance (in computation and storage) among processes. Nevertheless, we plan to investigate what forms of sparsity can be incorporated in the developed notation, thereby supporting a broader set of applications.

4.2 Additional Families of Algorithms

In this work, we focused on the stationary family of algorithms; however, so-called “3D” families of algorithms [1, 23, 26] for matrix-matrix multiplication have benefits in certain settings and generalizations have been incorporated into other related projects. We will investigate how to incorporate that family, and other potential families, of algorithms in the developed notation as part of future research.

4.3 Additional Data Distributions

We focused on formalizing elemental-cyclic distributions. Other projects rely on different forms of Cartesian distributions [4, 7] and other applications as well. For instance, some chemistry methods rely on distributions blocked distributions [5]. We plan to investigate how different Cartesian distributions can be incorporated into our notation.

4.4 Additional Tensor Operations

The application focused on in this dissertation arises from computational chemistry and the operation focused on was the tensor contraction. Tensor operations, such as factorizations, have also been shown to arise in the area of data analysis [11, 2, 3]. As part of future research,

we plan to investigate how to incorporate these operations into the defined notation and formalism.

4.5 Aiding Automated Tools

Blocking computations into subproblems can significantly reduce the amount of workspace required for computation. Encoding this knowledge so that automated tools can effectively reason about such aspects can aid them in determining the best implementation. We plan on investigating other useful knowledge to encode for automated systems as well as how to encode them.

5 Conclusion

In this work, we developed a notation that encodes information for distributing higher-order tensors on multidimensional processing meshes based on elemental-cyclic data distributions. In doing so, a systematic procedure for deriving algorithms is exposed that facilitates the automatic generation of high-performance implementations for a series of tensor contractions. This notation extends the stationary family of algorithms, shown to be effective for matrix-matrix multiplication, to tensor contractions. Experiments performed on the IBM Blue Gene/Q and Cray XC30 architectures show that implementations generated using the ideas developed in this document can improve over other state-of-the-art methods both in terms of performance and storage requirements.

References

- [1] R.C. AGARWAL, S. M. BALLE, F. G. GUSTAVSON, M. JOSHI, AND P. PALKAR, *A three-dimensional approach to parallel matrix multiplication*, IBM Journal of Research and Development, 39 (1995), pp. 39–5.
- [2] C. M. ANDERSEN AND R. BRO, *Practical aspects of PARAFAC modeling of fluorescence excitation-emission data*, Journal of Chemometrics, 17 (2003), pp. 200–215, [doi:10.1002/cem.790](#), [10.1002/cem.790](#).
- [3] BRETT W. BADER, MICHAEL W. BERRY, AND MURRAY BROWNE, *Discussion tracking in Enron email using PARAFAC*, in Survey of Text Mining II, Springer London, 2008, pp. 147–163, [doi:10.1007/978-1-84800-046-9_8](#).
- [4] G. BAUMGARTNER, A. AUER, D.E. BERNHOLDT, A. BIBIREATA, V. CHOPPELLA, D. COCIORVA, X GAO, R.J. HARRISON, S. HIRATA, S. KRISHNAMOORTHY, S. KRISHNAN, C. LAM, QINGDA LU, M. NOOIJEN, R.M. PITZER, J. RAMANUJAM, P. SADAYAPPAN, AND A. SIBIRYAKOV, *Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models*, Proceedings of the IEEE, 93 (2005), pp. 276–292, [doi:10.1109/JPROC.2004.840311](#).
- [5] NELSON H. F. BEEBE AND JAN LINDERBERG, *Simplifications in the generation and transformation of two-electron integrals in molecular calculations*, International Journal of Quantum Chemistry, 12 (1977), pp. 683–705, [doi:10.1002/qua.560120408](#).
- [6] JIŘÍ ČÍŽEK, *On the correlation problem in atomic and molecular systems. Calculation of wavefunction components in Ursell-type expansion using quantum-field theoretical methods*, The Journal of Chemical Physics, 45 (1966), pp. 4256–4266, [doi:10.1063/1.1727484](#).
- [7] ERIK DEUMENS, VICTOR F. LOTRICH, AJITH PERERA, MARK J. PONTON, BEVERLY A. SANDERS, AND RODNEY J. BARTLETT, *Software design of ACES III with the super instruction architecture*, Wiley Interdisciplinary Reviews: Computational Molecular Science, 1 (2011), pp. 895–901, [doi:10.1002/wcms.77](#), [10.1002/wcms.77](#).
- [8] T. HAZAN, S. POLAK, AND A. SHASHUA, *Sparse image coding using a 3D non-negative tensor factorization*, in ICCV 2005: Tenth IEEE International Conference on Computer Vision, 2005, pp. 50–57, [doi:10.1109/ICCV.2005.228](#).
- [9] T. HELGAKER, P. JØRGENSEN, AND J. OLSEN, *Molecular Electronic Structure Theory*, John Wiley & Sons, LTD, Chichester, 2000.
- [10] HENK A. L. KIERS, JOS M. F. TEN BERGE, AND ROBERTO ROCCI, *Uniqueness of three-mode factor models with sparse cores: The $3 \times 3 \times 3$ case*, Psychometrika, 62 (1997), pp. 349–374, [doi:10.1007/BF02294556](#).
- [11] TAMARA G. KOLDA AND BRETT W. BADER, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500, [doi:10.1137/07070111X](#), [arXiv:10.1137/07070111X](#).

- [12] STANISLAW A. KUCHARSKI AND RODNEY J. BARTLETT, *Recursive intermediate factorization and complete computational linearization of the coupled-cluster single, double, triple, and quadruple excitation equations*, Theoretica Chimica Acta, 80 (1991), pp. 387–405, doi:[10.1007/BF01117419](https://doi.org/10.1007/BF01117419), [10.1007/BF01117419](https://doi.org/10.1007/BF01117419).
- [13] CHUN-YUAN LIN, YEH-CHING CHUNG, AND JEN-SHIUH LIU, *Efficient data compression methods for multidimensional sparse array operations based on the EKMR scheme*, IEEE Transactions on Computers, 52 (2003), pp. 1640–1646, doi:[10.1109/TC.2003.1252859](https://doi.org/10.1109/TC.2003.1252859).
- [14] BRYAN MARKER, DON BATORY, AND ROBERT VAN DE GEIJN, *A case study in mechanically deriving dense linear algebra code*, International Journal of High Performance Computing Applications, 27 (2013), pp. 439–452.
- [15] BRYAN MARKER, DON BATORY, AND ROBERT VAN DE GEIJN, *Code generation and optimization of distributed-memory dense linear algebra kernels*, Procedia Computer Science, 18 (2013), pp. 1282–1291, doi:[10.1016/j.procs.2013.05.295](https://doi.org/10.1016/j.procs.2013.05.295).
- [16] DEVIN A. MATTHEWS, *Non-orthogonal Spin-adaptation and Applications to Coupled Cluster up to Quadruple Excitations*, PhD thesis, The University of Texas at Austin, August 2014.
- [17] JOZEF NOGA AND RODNEY J. BARTLETT, *The full CCSDT model for molecular electronic structure*, The Journal of Chemical Physics, 86 (1987), pp. 7041–7050, doi:[10.1063/1.452353](https://doi.org/10.1063/1.452353).
- [18] SCOTT PARKER, *BG/Q architecture*. Argonne National Laboratory, May 2013, https://www.alcf.anl.gov/files/bgq-arch_0.pdf.
- [19] GEORGE D. PURVIS AND RODNEY J. BARTLETT, *A full coupled-cluster singles and doubles model: The inclusion of disconnected triples*, The Journal of Chemical Physics, 76 (1982), pp. 1910–1918, doi:[10.1063/1.443164](https://doi.org/10.1063/1.443164).
- [20] S. RAJBHANDARI, A. NIKAM, PAI-WEI LAI, K. STOCK, S. KRISHNAMOORTHY, AND P. SADAYAPPAN, *A communication-optimal framework for contracting distributed tensors*, in SC14: International Conference for High Performance Computing, Networking, Storage and Analysis, 2014, pp. 375–386, doi:[10.1109/SC.2014.36](https://doi.org/10.1109/SC.2014.36).
- [21] MARTIN D. SCHATZ, *Distributed Tensor Computations: Formalizing Distributions, Redistributions, and Algorithm Derivation*, PhD thesis, The University of Texas at Austin, 2015.
- [22] MARTIN D. SCHATZ, TZE MENG LOW, ROBERT A. VAN DE GEIJN, AND TAMARA G. KOLDA, *Exploiting symmetry in tensors for high performance: Multiplication with symmetric tensors*, SIAM Journal on Scientific Computing, 36 (2014), pp. C453–C479, doi:[10.1137/130907215](https://doi.org/10.1137/130907215).

- [23] MARTIN D. SCHATZ, JACK POULSON, AND ROBERT VAN DE GEIJN, *Parallel matrix multiplication: 2D and 3D*, FLAME Working Note #62 TR-12-13, The University of Texas at Austin, Department of Computer Sciences, 2012.
- [24] ISIAH SHAVITT, *The method of configuration interaction*, in Methods of Electronic Structure Theory, vol. 3 of Modern Theoretical Chemistry, Springer US, 1977, pp. 189–275, doi:[10.1007/978-1-4757-0887-5_6](https://doi.org/10.1007/978-1-4757-0887-5_6).
- [25] ISIAH SHAVITT AND RODNEY J. BARTLETT, *Many-Body Methods in Chemistry and Physics*, Cambridge University Press, 2009, doi:[10.1017/CB09780511596834](https://doi.org/10.1017/CB09780511596834).
- [26] EDGAR SOLOMONIK AND JAMES DEMMEL, *Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms*, in Euro-Par 2011 Parallel Processing, vol. 6853 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 90–109, doi:[10.1007/978-3-642-23397-5_10](https://doi.org/10.1007/978-3-642-23397-5_10), [10.1007/978-3-642-23397-5_10](https://doi.org/10.1007/978-3-642-23397-5_10).
- [27] E. SOLOMONIK, D. MATTHEWS, J.R. HAMMOND, AND J. DEMMEL, *Cyclops tensor framework: Reducing communication and eliminating load imbalance in massively parallel contractions*, in 2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS), 2013, pp. 813–824, doi:[10.1109/IPDPS.2013.112](https://doi.org/10.1109/IPDPS.2013.112).
- [28] EDGAR SOLOMONIK, DEVIN MATTHEWS, JEFF R HAMMOND, JOHN F STANTON, AND JAMES DEMMEL, *A massively parallel tensor contraction framework for coupled-cluster computations*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3176–3190.
- [29] M. VALIEV, E. J. BYLASKA, N. GOVIND, K. KOWALSKI, T. P. STRAATSMA, H. J. J. VAN DAM, D. WANG, J. NIEPLOCHA, E. APRA, T. L. WINDUS, AND W. A. DE JONG, *NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations*, Computer Physics Communications, 181 (2010), pp. 1477–1489, doi:[10.1016/j.cpc.2010.04.018](https://doi.org/10.1016/j.cpc.2010.04.018).
- [30] *NERSC Edison configuration.* <https://www.nersc.gov/users/computational-systems/edison/configuration/>.

DISTRIBUTION:

- 1 MS 0899 Technical Library, 8944 (electronic)
- 1 MS 0123 D. Chavez, LDRD Office, 1011